A Case-study for the Semantic Analysis of Sentences in Coq

Line Jakubiec

A Case-study for the Semantic Analysis of Sentences in Coq

Line Jakubiec-Jamet

Laboratoire d'Informatique Fondamentale de Marseille (L.I.F-UMR7279)
Aix-Marseille Université
Faculté des Sciences de Luminy
163, Avenue de Luminy - Case 901
13288 Marseille Cédex 9 - France
Line.Jakubiec@lif.univ-mrs.fr

Abstract. This paper presents a case-study devoted to the formalization of sentence frames in the Coq system. Therefore, we instanciate these frames for performing a semantic analysis of simple sentences. In particular, we rely on a hierarchy of types for type-checking the conceptual well-formedness of sentences. To do so, we investigate how to exploit the particular features of the Coq type system in order to take advantage of this elegant unifying framework for encoding the syntax-semantics interface and then we show how to improve our approach for combining it with linguistic resources distributed.

Keywords: natural language processing, Coq system, semantics, conceptual analysis

1 Introduction

During the last fifteen years, the use of type theoretic methods for describing natural langage syntax and semantics has gained more and more popularity, resulting in the development of software relying on these methods. On the one hand, these type theoretic methods can be constantly improved with advances in the field of logic. On the other hand, they depend on the developement of linguistic resources such as lexicons or dictionaries. In this context and from our point of view, the challenge for natural langage processing is twofold : first, the frameworks dedicated to linguistic analysis have to focus on the syntax-semantics interface ; secondly, they have to combine linguistic resources and natural language processing programs. Among the most significant achievements, let us mentionned the works on categorial grammars which provide the integration of syntax and semantics in the same framework as it is described in [19] and in [29]. Moreover, categorial grammars are lexicalized that means that all items in the lexicon are typed. Thereby, although they describe syntactical rules, they also preserve the compositional aspect of Montague semantics [18]. The most well-known categorial grammars are those based on Lambek-Calculus [16]. Due to the Curry-Howard isomorphism, typed terms are proofs in logic which includes

Lambek-Calculus[1]. Although many studies have already showed that it is natural to associate a syntactic term to a semantic type and reciprocally [28] [22] [23], our approach implemented in Coq, the Calculus of Inductive Constructions, aims to focus on the generality of definitions leading to reusable methodologies dedicated to pragmatic[2] semantic analysis. In Coq, few investigations have been performed for natural language processing. [8] developed an algorithm that produces natural language sentences from proofs described in a mathematical language. Later, for the case of categorial grammars, [1] gave a Coq formalization of the Lambek-Calculus and of an extension as multimodal grammars, and proved in particular several theorems as completeness and consistency of multimodal logic.

However, we are not aware of any work that straightforwardly used specific features of Coq language, for linguistic analysis. This paper presents on a case-study, the capabilities of Coq for the determination of a sentence's conceptual well-formedness from general representations of sentences. Let us remark that the conceptual analysis is part of the traditionnal semantic analysis which takes into account the contextual analysis as well. Actually, the former determines the compatibility of the elements from which the sentence is composed (compositionality of words), while the latter studies the combination of sentences with each others (texts). The study is inspired by concepts developped in a prototype, Illico [25], realized in the research team TALEP (LIF)[3]. In Illico, the conceptual analysis is based on a first-order typed λ-calculus which is implemented in Prolog. Then, these Prolog rules are used to define relations that represent sentences. Moreover, the conceptual analysis is encoded according to conceptual types that are used to determine the compatibility of the words in a sentence. Types are hierarchically organized and they describe an ontology[4] itself implemented as Prolog rules.

This paper proposes a more straightforward approach and the relevance of our encoding in Coq can be summarized as follows :

1. define formal models for representing sentences by taking advantage of the Coq type system and its particularly rich language (polymorphism, higher-order logic, coercion mechanism, module system),

[1] Note that there are new approaches that extend Lambek-Calculus, in particular in linear logic because it provides an efficient representation of proofs [9] [27] [17] and specific tools have been developed in this field as for example [20].

[2] The semantic analysis presented in this paper is pragmatic is the sense that, it plans to use linguistic resources mentionned in the section 5 in order to obtain a tool used by linguists.

[3] Traitement Automatique du Langage Ecrit et Parlé (Laboratoire d'Informatique Fondamentale de Marseille)

[4] Ontologies are widespread in the process of improving sentence's analysis. Many studies are based on the development of ontologies specially designed for managing the conceptual knowledge [2] [4] [24]. The advantages of using ontologies in natural language processing are : on the one hand, they provide a way to represent semantics of a given domain on which sentences have to be analysed ; on the other hand, it is possible to reuse or reorganize the knowledge depicted into the ontology.

2. propose natural and general specifications of sentences that can be checked for a conceptual analysis based on types,
3. directly encode typed λ-terms without first specifying λ-calculus in a language and then, implement representations of sentences in the specified λ-calculus,
4. take advantage of type-checking algorithms involved into the Coq system.

The paper is organized as follows. Section 2 briefly introduces Coq by focusing on used aspects. Section 3 deals with a formalization of the underlying ontology and with a semantic representation of simple sentences. In Section 4, we generalize our approach by specifying generic models for other sentences and we show how to use them. We mention in section 5 some tools and resources for improving our work. Then, on the conclusion, we further discuss the case-study and we highlight our perspectives.

2 An Overview of Coq

The Coq system [5] is a specification and proof system developed in the LogiCal project at Laboratoire de Recherche en Informatique (CNRS and University of Paris-Sud) and LIX (INRIA-Futurs and Ecole Polytechnique). Coq's language relies on a higher-order typed λ-calculus, the Calculus of Constructions [6] [7] enriched with inductive and co-inductive definitions [26] [12]. Coq's logic is a constructive logic and it is based on the *propositions-as-types* correspondence, the Curry-Howard isomorphism, that states a proposition is a type and a proof is a term inhabiting this type. This correspondence provides an elegant unifying framework where type-checking is proof-checking. The Coq system is tactic oriented and it allows to interactively develop proofs. The system is organized around a small kernel (the theory) extended by libraries. Moreover, it includes many user contributions.
Coq developments can be splitted into various parameterized modules. Thus, several developments can share modules that, being compiled once and for all, are loaded fast. Moreover, sections allow to organise modules in a structured way. In Coq, any user's term must be classified according to a type. There are two sorts of types : logical propositions are of sort $Prop$ and mathematical collections are of sort Set[5]. Polymorphic terms are parameterized with respect to terms of sort $Prop$ or Set. By defining them into a section, one can obtain reusable developments in which generic specifications has been already typed-checked. However, when instanciating these abstract specifications, types can be cumbersome. The implicit parameter mechanism allows to automatically infer arguments from the definition's context.
Moreover, Coq terms are organised according to a type hierarchy and they can be typed using the coercion subtyping system. This latter corresponds to an inheritance graph mechanism that allows to inject Coq hierarchy typed terms into

[5] Actually, this distinction is not necessary but it makes the system less confusing for the user. However, it is significant when extracting programs from proofs (a mechanism relying on the constructive aspect of Coq's logic). But this feature is not used here. For more details, one can refer to [5].

another hierarchy's type. Technically, a term of type t is also of type t_1 (where t and t_1 are of types $Prop$ or Set for example) if there exists a coercion between t_1 and t, defined in Coq as :

```
Coercion c : t1 >-> t.
```

The above declaration expresses the construction denoted by c as a coercion between $t1$ and t. Roughly speaking and for the need of the study, the coercion c indicates that t_1 is a subtype of t.

3 Sentence's Conceptual Analysis

3.1 The Starting Point : Illico's Conceptual Analysis

Illico is a generic software tool designed for analysis and synthesis of natural language [25]. It provides tools and formalisms to encode and to analyse lexical, syntactic and semantic knowledge. This latter allows the determination of a sentence's conceptual and contextual well-formedness. To sum up, Illico is able to perform any of the following tasks :

1. analyze sentences and display the representations associated with each one of them at the syntactic and semantic levels ;
2. detect during analysis, lexically, syntactically or semantically unexpected words ;
3. guide the composition of sentences, by displaying possible continuations for the user's sentence.

From a computational point of view, the Illico semantic analysis includes the following aspects :

1. Formal semantic representation : once the lexical and syntactical levels of a natural language expression are completed, then Illico automatically generates a semantic representation of this expression. This latter is encoded from composition rules, using first-order typed λ-calculus.
2. Conceptual validation : then, the typed λ-expression must be conceptually verified. If it describes a compatible situation with the underlying ontology, the conceptual well-formedness is established.

Technically, as it has been mentionned in Section 1, these two considerations are managed with Prolog rules. Semantic representations are λ-expressions encoded in Prolog and they are defined from a set of semantic labels. These latter corresponds to two basic types : i (that stands for agent) and o (that stands for boolean expressions or relations).
For example, a verbal group is labelled $< i, o >$ because it needs an agent (the subject) and an evaluation (the application of the subject to the verb). The expression $verbal\_group = \lambda x\ (name\_group\ (verb\ x))$ where $name\_group$ and $verb$ are respectively labelled $<< i, o >, o >$ and $< i, < i, o >>$, represents the

semantic rule to be conceptually verified[6]. Then, the conceptual validation is based on a conceptual tree as it is described hereunder.

3.2 Ontology as Concept Hierarchy

This section is devoted to the presentation of the ontology used to determine the well-formedness of sentences. Let us note that the "concepts as types" representation is largely used in the knowledge representation since it provides an appropriate interface for semantic processing.
Figure 1 presents a conceptual hierarchy which organises concepts according a world we want to refer for our study. The verification of the sentence's conceptual well-formedness will be based on this hierarchy. It describes a simple world from which it is possible to analyse the meaning of sentences. This world is organized from *animate* and *inanimate* notions. For example, an *animal* is classified into the *animate* concept while a *car* is *inanimate* and can be considered as *concrete* as well. In this tree, each node is labelled by a conceptual information that can be specified as a type. Thus, for organizing this information, it is natural to consider the subtyping principle. In typed definitions, a subtype may appear wherever an element of the super type is expected. For example, in our verb's semantic representation (Section 3.3), a type t_1 is compatible with a type t, if t_1 is a subtype of t. Consequently, a semantic representation parameterized with t will be valid for parameters of type t_1 and for all those of the lower part.

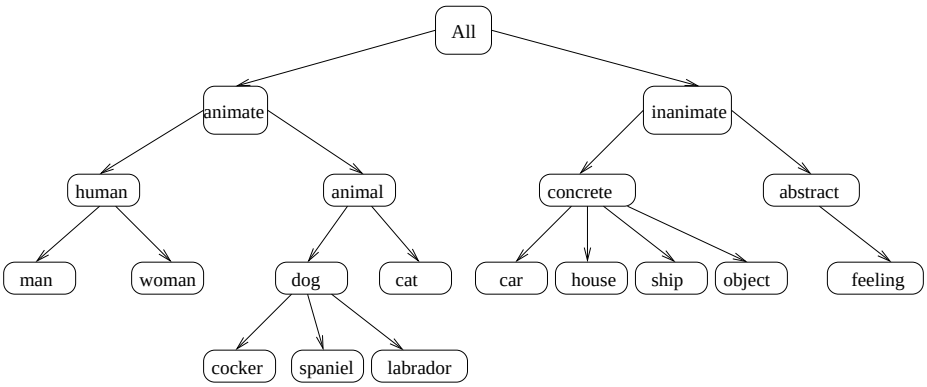

Fig. 1. A hierarchy of conceptual types

In Coq, we declare the conceptual types (*all*, *animate*...) as logical propositions. Then, we use the coercion mechanism for describing the relations between types, as follows :

[6] Moreover, these semantic rules are associated to syntactic rules involved in the grammar of Illico named GNF (Grammaire Noyau du Français).

```
Coercion animate_is_all    : animate >-> all.
Coercion animal_is_animate : animal >-> animate.
Coercion dog_is_animal     : dog >-> animal)
Coercion cat_is_animal     : cat >-> animal)
      ......
```

Each coercion creates a path between two nodes of the tree. The whole list of coercions is ordered and it defines the conceptual tree depicted in Figure 1. Coq detects ambiguous paths during the creation of the tree and, it verifies the *uniform inheritance condition* because at most one path must be declared between two nodes. Let us remark that, in Coq, the conceptual tree is straightforwardly implemented, in a natural way. Therefore, Coq automatically verifies that the hierarchy of types is well-formed. Moreover, the conceptual analysis will be parametrizable by this kind of hierarchy : in general, the ontology depends on the field under consideration and it is interesting to be able to change the ontology according the needs. Just for comparaison, the corresponding specification in Prolog is given hereunder :

```
domain:all(all.v24457) ->  ;
domain:animate(all.animate.v24459) ->  ;
domain:animal(all.animate.animal.v24460) ->  ;
domain:dog(all.animate.animal.dog.v24461) ->  ;
domain:cat(all.animate.animal.cat.v24461) ->  ;
```

It shows all the necessary paths from the root *all*. This Prolog specification is generated from the Illico interface : a parser analyses rules given by the users (as for example $all = \{animate, inanimate\}$) and constraints are introduced (into the parser) for avoiding incompatibilities in the tree ; in Coq, this processing can be directly used by means of the coercion mechanism. Later, in Section 5, we propose a tool that generates the hierarchy from a graphical interface for alleviating the creation of ontologies in Coq.

3.3 Coq Semantic Representation of Sentences

This section proposes a semantic formalization of simple sentences. Since a sentence is composed of elements that must be compatible with each other, we first deal with their representation and their types. In particular, we focus on the verb's description because the sentence's representation is specified according to the verb's domain of use. Then we describe a generic model for sentences which involve a verb and its subject. Finally, we show, by instanciation of the model, how sentences are semantically represented.
Therefore, the verb has a central role in the sentence, as it has been developed in the Tesnière's linguistic theory [31] which defines the valence of verbs. Let us mention that, this characterisation of sentences have been widely used in the dependency grammars as it is described in [15].

Lexicon of Verbs For typing the verb's representation, we use the conceptual types described in Figure 1. For each verb, we have to choose the best label (best

for the user who build the lexicon that is to say the most likely concept for the domain under consideration) which, in general, corresponds to the lower type in the hierarchy. For example, the verb *to bark* can reasonably be used for all the dogs. So, it is declared in the lexicon as a logical proposition by the unary predicate *bark* as follows :

```
Parameter bark : dog -> Prop.
```

Let us note that verbs can have different meanings and so, different lexical entries have to be considered. In [21], the authors deal with this aspect on a study formalized in categorial grammars. They introduce constraints which change the interpretation of verbs by means of specific functions that change the type of terms. Our approach is different because once the hierarchy of types is defined, the verbs have to be described on these types. So, we can need to consider several declarations in Coq for the same verb. For example in french, the verb *bark* can be used for humans : *Paul barks.* (in the sense that Paul inveighs against someone or something) is an acceptable sentence. Here, we introduce a new input in our lexicon as :

```
Parameter bark2 : human -> Prop.
```

This is not really cumbersome because dictionnaries of verbs are built from all the possible situations (as it is described in the Section 5) and then, the Coq representation is very concise.

In Illico, the lexicon entry has to be duplicated as well (if needed). Let us remark that each input of the lexicon is specified by two following rules as, for example :

```
lexicon(verb0,<to_bark>,<is_barking>,...) -> ;
semantic.txt:verb_0 = <i,o> ;
```

where informations as lexical category ($verb0$), lemmatized form ($< to\_bark >$), used form ($< is\_barking >$) and other features appear. These characteristics are used for syntactic verification as well, but it is not the focus here.

Sentences as logical expressions Let us now consider the sentence : *A dog is barking.* It can be represented by the logical expression : $\exists x, bark(x) \wedge is\_dog(x)$ where the unary predicate is_dog characterizes the semantics of the subject *dog*. The following predicates introduce in Coq semantic representation for some agents used later in the paper :

```
Parameter is_animate : all -> Prop.
Parameter is_animal  : animate -> Prop.
Parameter is_human   : animate -> Prop.
Parameter is_dog     : animal -> Prop.
Parameter is_cocker  : dog -> Prop.
```

The domain of definition is more general than the agent itself for avoiding dead end situation. It is a choice made in the Illico conception. For example, suppose the negative question : *Who is not an animal?* If is_animal is defined on $animal$, we can write the expression $\exists x, \neg is\_animal(x)$ (where x implicitly is of

type *animal* and x must be instanciated to obtain a final well-typed representation). But any x will exist because no conceptual verification will allow the final representation (x must be of type *animal* or one of the subtypes of *animal*). If is_animal is defined on *animate*, x will exist and can be, for example, a parameter h of type *human*, because the expression $\neg is\_animal(h)$ is valid.
Finally, for a first approach, the sentence : *A dog is barking* merely can be represented in Coq as follows :

```
Definition a_dog_is_barking :=  exists x, bark(x) /\ is_dog(x).
```

where the implicit argument mechanism automatically synthesizes the type of x as *dog* (from the first predicate of the definition).

Towards Generic Models Up to now, the kind of sentences we study is composed of a verb and a subject. In this part, we show how to generalize this kind of sentences by specifying a general frame in Coq that states : $\exists x, verb0(x) \wedge is\_something(x)$, where $verb0$ (that stands for the verb) and $is\_something$ (that stands for the subject) are polymorphic predicates respectively parameterized on $A1$ and A of sort *Prop*, with $A1$ subtype of A (to ensure the compatibility between words). The complete specification in Coq is given below, inside a section :

```
Section General_frame_v0.

  Variables (A A1:Prop)          (** Local parameters of the section **)
            (A1A : A1 -> A).

  Coercion A1A : A1 >-> A.       (** Declaration of the subtype **)

  Variables (verb0 : A1 -> Prop) (** Declaration of the predicates **)
            (is_something : A -> Prop).

 (** Definition of the generic model **)
  Definition frame_verb0 := exists c, verb0 c /\ is_something c.

End General_frame_v0.
```

In the definition $frame\_verb0$, c is implicitly of type $A1$ and the coercion $A1A$ which converts the type $A1$ to A is implicitly applied on c into $is\_something(c)$. Outside the section, the local context of the definition is discharged. This means that A, $A1$, $A1A$, $verb0$ and $is\_something$ appear as parameters of the definition $frame\_verb0$.
So, $frame\_verb0$ depends on two types, on a coercion which states a subtyping relation and on two predicates which respectively stand for a verb and a subject. It is a generic representation for this kind of simple sentences due to polymorphism (from the parameters A and $A1$) and higher-order (from the $verb0$ and $is\_something$ predicates).

Type-checking is Well-formedness Checking By instanciation of the generic model $frame\_verb0$, we can define the semantic representation of the sentence *A dog is barking* as :

```
Definition a_dog_is_barking := (frame_verb0 dog_is_animal bark is_dog).
```

The instanciation of the parameters A and $A1$ can be omitted due to the implicit synthesis. The coercion dog_is_animal instanciates $A1A$ in the generic model ; $bark$, which is defined on dog, instanciates $verb0$ and is_dog, which is defined on $animal$, instanciates $is\_something$. So, the compatibility of words is ensured by the coercion that states dog is a subtype of $animal$.

In a similar way, the sentence *A cocker is barking* is formalized as :

```
Definition a_cocker_is_barking :=
           (frame_verb0 cocker_is_dog bark is_cocker).
```

But the checking of *A cat is barking* :

```
Definition a_cat_is_barking := (frame_verb0 cat_is_animal bark is_cat).
```

is rejected by the system because the term $bark$ has type $dog \rightarrow Prop$, while it is expected from cat_is_animal (the coercion that states cat is a subtype of $animal$) to have type $cat \rightarrow Prop$.
This encoding leads to simple conceptual representation of sentences. Coq performs the type-checking of these instanciations and so, it establishes the sentence's well-formedness.

4 Generic Models Based on Verb's use

In this section, we propose two other generic frames for representing the sentences. The first one describes sentences which are composed of a verb, a subject and a complement. The second frame defines sentences where the parameters of the verb depends on each others. This latter is interesting is the case-study because it emphasizes the dependent aspect of typing for natural language processing.

Let us consider the sentence *A woman likes cars*. In this case, the verb *to like* may be of type $animate \rightarrow inanimate \rightarrow Prop$, where $animate$ stands for the type of the subject and $inanimate$ for the type of the complement[7]. Similarly to the representation given in Section 3.3, we can specify :

```
Definition a_woman_likes_cars :=
           (frame_verb1 woman_is_human car_is_concrete
                        like is_woman is_car).
```

where the model $frame\_verb1$ is obtained from the generic definition depicted below :

[7] But, more generally, the type of the verb *to like* in the lexicon is $animate \rightarrow all \rightarrow Prop$ because the type of the complement must be as general as possible.

```
Section General_frame_v1.

  Variables (A A1 B B1:Prop)
            (A1A : A1 -> A)
            (B1B : B1 -> B).

  Coercion A1A : A1 >-> A.
  Coercion B1B : B1 >-> B.


  Variables (verb1 : A1 -> B1 -> Prop)
            (is_something1 : A -> Prop)
            (is_something2 : B -> Prop).

  Definition frame_verb1 :=
      exists x, exists y, verb1 x y /\
                          is_something1 x /\ is_something2 y.

End General_frame_v1.
```

As in the previous frame, coercions has been defined between $A1$ and A and between $B1$ and B. The predicates $is\_something1$ and $is\_something2$ respectively stand for the subject and the complement. So the instanciation of $frame\_verb1$ is realized using the two coercions $woman\_is\_human$ and $car\_is\_concrete$, the verb $like$ and the predicates is_woman and is_car.

The second model describes a particular case where sentences are composed of a verb, a subject and a complement as well, but the typing of the verb is more binding. For example, let us consider the verb $confuse$. Only a human can $confuse$ two things that must represent the same concept in the hierarchy (for instance, a man confuse two dog breeds, two particular cars and so on ; but he cannot confuse a car and a dog). So, the type of the generic relation ($verb2$ in the hereunder definition) that stands for $confuse$ is $human \rightarrow A \rightarrow A \rightarrow Prop$ where A gives the general concept to be used but the two subtypes of A can be different althought they are from the same concept. This is specified in the next Coq definition inside a section :

```
Section General_frame_v2.

  Variable A A1 A2 : Prop.
  Variable A1A : A1->A.
  Coercion A1A : A1>->A.
  Variable A2A : A2->A.
  Coercion A2A : A2>->A.

  Variables (verb2 : human-> A ->  A -> Prop)
            (is_something1 : A1 -> Prop)
            (is_something2 : A2 -> Prop).
```

```
  Definition frame_verb2 : Prop :=
     exists h:human, exists a1:A1, exists a2 :A2,
     verb2 h a1 a2 /\ is_human h /\ is_something1 a1 /\ is_something2 a2.

End General_frame_v2.
```

The two types from A are $A1$ and $A2$; the variable $verb2$ sets that the first argument is a human while the following are from A.
As already described, due to implicit synthesis, the application ($verb2$ h $a1$ $a2$) is actually ($verb2$ h ($A1A$ $a1$) ($A2A$ $a2$)), for generating the type A from $A1$ and from $A2$. The instanciation of this frame is similar to the previous paragraphs and it is not given here.

5 Tools and resources

This section is devoted to the presentation of practical tools for faciliting the use of our work in Coq. Really, the interface presented to the user has to hide the logical aspects of the Coq specifications which can be cumbersome for a linguistic analysis. The Figure 2 gives an overview of the application based on the Coq type-checking presented in the paper. It is not yet fully implemented but it allows to explain the motivations of our work.

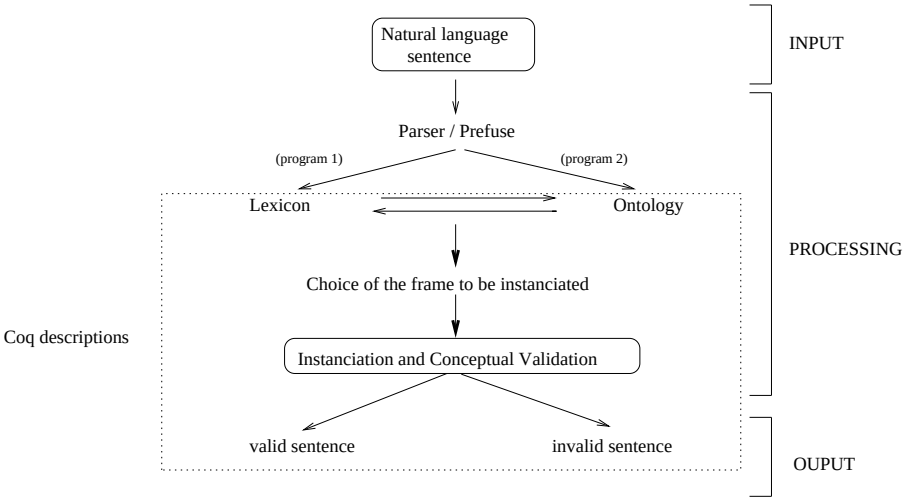

Fig. 2. Overview of the application based on a Coq analysis

The application takes as input a natural language sentence. A parser implemented in Java allows to interactively tag verbs the user wants to classify according an ontology (as for example, the Figure 1). This classification ($program1$ in

the above figure) provides a lexicon of verbs in Coq (for instance, *bark* is tagged as $dog \rightarrow Prop$ depending on the choice of the user). The Coq description of the ontology can be automatically obtained by the *program*2 [3] in the figure. This program uses the Prefuse toolkit [14]. Prefuse allows the user to graphically design its ontology without specifying Coq descriptions ; so *program*2 generates the Coq file that describes the coercion between types and also the semantic predicates corresponding to the ontology ($is\_dog : animal \rightarrow Prop$ for instance). Then the user chooses the Coq generic frame to be instanciated and the conceptual validation is performed as it has been depicted in this paper. If the representation of the sentence could be type-checked into Coq, the application outputs that the sentence is valid ; otherwise, it returns that the sentence is not valid.

This application is motivated from the fact that there are few resources in french for having semantics on words. The classification mechanism seems to be interesting for our study because it allows to abstract properties for classes of verbs. Several linguistic resources rely on this aspect. On the one hand, there are dictionaries that provide a precise explanation of words : in [10], verbs are described as semantics classes whose scope is defined by syntax. These classes are generic and each of them gathers properties of verbs. For example, there is the class dedicated to the communication verbs. This latter is divided into 4 semantic categories (for french language) :

1. human, animal (to shout, to speak)
2. human (to say something)
3. human (to show)
4. figurative sense

Then these categories are subdivided into syntactical sub-classes which describe the use cases of verbs (the verb can be used with a subject and a complement, it can be transitive or intransitive and so on).

On the other hand, lexicons have been developed from dictionaries and [10] have been used in several implementations as in [30], [11] and [13]. The study of these resources for the development of our application should be useful because they provide many linguistic descriptions not taken into account in our case-study. So they may be reused and connected to our application.

6 Conclusion and Perspectives

The work presented in this paper aims at studying the capabilities of the Coq system, in the field of semantic representation and conceptual analysis for natural language processing. The relevance of our encoding has been motivated by the particular features of the Coq system. It provides an unifying framework with a rich type system that allows to straightforwardly specify the underlying conceptual tree as well as to analyse semantic representations. In this paper we have proposed :

1. a way for describing ontologies in Coq,
2. several reusable sentence's semantic representations,
3. a sentence's conceptual analysis by instanciation process.

In particular, for this analysis, we focused on several aspects of the Coq system :

1. polymorphism : the generic models of sentences are parametrized by types and they can be reused for specifying specific sentences.
2. higher-order : it allows to take as parameters relations and so it provides a good framework for developing general definitions. From these definitions, specific sentences are derived by instanciation.
3. modularity : the development is split into several sections. This contributes to lisibility and it allows an easy reuse of libraries.
4. coercion mechanism : the hierarchy of types is easily described in Coq and it is a good way for knowledge representation based on types.
5. implicit parameters : the implicit synthesis of some parameters greatly improves the readability of the definitions.

The case study proposed in this paper is being extended to several other modules. In particular, it requires the addition of the following :

1. extension of the verb's lexicon : the lexicon takes into account around 50 verbs. We plan to use the developments of LVF mentionned in the Section 5 for improving the semantic analysis (by specifying classes of verbs and general models that characterize the uses of verbs). This study will allow to integrate syntactical rules for introducing more linguistic information in our representations. From this specification, formal properties about sentence's representations could be established in order to validate linguistic rules in Coq.
2. representation of other sentences : once the lexicon will be extended, it will be possible to describe other generic models of sentences.
3. contextual analysis : for completing the semantic analysis, contextual representation will be necessary. This will allow to analyse texts and not just sentences.

Finally, the application depicted in the Figure 2 should be completed by adding more automatic processing : firstly, in the parser, the tagging of verbs has to be improved. This can be done using an underlying lexicon which includes linguistic properties about verbs (conjugaison patterns, lemmatization for example). Secondly, the interface between the lexicon and the ontology has to be developped. This process involves the programming of an interface that allows to manipulate verbs and concepts according to the domain of the sentences to be analysed. Thirdly, it should be relevant to give more explanation about the output when an invalid sentence has been detected. This is possible by retrieving from the Coq system the information about typing, in order to propose for the user, some accepted constructions of sentences.

References

1. Anoun, H.: Reasoning on Multimodal Logic with the Calculus of Inductive Constructions. In: Logic for Programming Artificial Intelligence Reasoning (2005)
2. Asher, N.: Lexical Meaning in Context. a Web of Words. In: Cambridge University Press (2011)
3. Azzabi, M., Giamarchi, G., Houillon, C.: Description et analyse d'arbres à l'aide de Prefuse (2007)
4. Bottazzi, E., Ferrario, R.: Preliminaries to a DOLCE Ontology of Organizations. In: International Journal of Business Process Integration and Management (2006)
5. Coq Development Team, INRIA: The Coq Proof Assistant. Reference manual, v8.3 (2004-2010)
6. Coquand, T.: Une Théorie des Constructions. Ph.D. thesis, Université Paris 7 (Janvier 1989)
7. Coquand, T., Huet, G.: Constructions : A Higher Order Proof System for Mechanizing Mathematics. EUROCAL 85, Linz Springer-Verlag LNCS 203 (1985)
8. Coscoy, Y.: Explication textuelles de preuves pour le calcul des constructions inductives. Thèse d'université, Université de Nice-Sophia-Antipolis (2000)
9. De Groote, P., Rétoré, C.: Semantic Readings of Proof Nets. In: Kruijff G., Morrill G., O.D. (ed.) Formal Grammar. pp. 57–70. FoLLI (1996)
10. Dubois, J., Dubois-Charlier, F.: Les verbes français. Larousse-Bordas (1997)
11. François, J., Le Pesant, D., Leeman, D.: Classements syntactico-sémantiques des verbes français. Langue française, vol. 153 (2007)
12. Giménez, E.: Un calcul de constructions infinies et son application à la vérification de systèmes communicants. Ph.D. thesis, Ecole Normale Superieure de Lyon (1996)
13. Hadouche, F., Lapalme, G.: Une version électronique du LVF comparée avec d'autres ressources lexicales. Langages, vol. 179-180, pp. 193–220 (2010)
14. Heer, J., Card, S., Landay, J.: Prefuse: a Toolkit for Interactive Information Visualization. In: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 421–430 (2005)
15. Kahane, S.: Grammaires de dépendance formelles et théorie sens-texte (2001)
16. Lambek, J.: The Mathematics of Sentence Structure. vol. 65, pp. 154–169 (1958)
17. Lecomte, A.: Grammaire et théorie de la preuve : une introduction. In: Traitement automatique des langues. vol. 37, pp. 1–38 (1996)
18. Montague, R.: The Collected Papers of Richard Montague. In: Thomason, R. (ed.) Yale University Press (1974)
19. Moortgat, M.: Categorial Type Logics. In: Benthem, J., Meulen, A.T. (eds.) Handbook of Logic and Language. pp. 93–177. North-Holland Elsevier-Amsterdam (1996)
20. Moot, R.: A short Introduction to Grail. In: Areces, C., Rijke, M. (eds.) Proceedings of Methods for Modalities (2001)
21. Moot, R., Prévot, L., Rétoré, C.: A Discursive Analysis of Itineraries in an Historical and Regional Corpus of Travels. In: Constraints in discourse (2011)
22. Moot, R., Rétoré, C.: The Logic of Categorial Grammars. vol. 6850. LNCS (2011)
23. Muskens, R.: Type-logical Semantics. In: Craig, E. (ed.) Routledge Encyclopedia of Philosophy Online. Routledge (2011)

24. Oltramari, A., Gangemi, A., Guarino, N., Masolo, C.: Restructuring WordNet's Top-level: The OntoClean Approach. In: Simov, K. (ed.) Workshop Proceedings of OntoLex'2, Ontologies and Lexical Knowledge Bases. pp. 17–26 (May 2002)
25. Pasero, R., Sabatier, P.: Illico. Tech. rep., LIF (2003)
26. Paulin-Mohring, C.: Inductive Definitions in the System Coq - Rules and Properties. In: Bezem, M., Groote, J.F. (eds.) Proceedings of the conference Typed Lambda Calculi and Applications. No. 664 in Lecture Notes in Computer Science (1993), lIP research report 92-49
27. Perrier, G.: Labelled Proof Nets for the Syntax and Semantics of Natural Languages. In: Logic Journal of the IGPL. vol. 7, pp. 629–654 (1999)
28. Ranta, A.: GF: a Multilingual Grammar Formalism. In: Language and Linguistics Compass. vol. 3 (2009)
29. Retoré, C.: Systèmes déductifs et traitement des langues, un panorama des grammaires catégorielles. TSI, vol. 20, pp. 301–336 (2001)
30. Silberztein, M.: La formalisation du dictionnaire LVF avec Nooj et ses applications pour l'analyse automatique de corpus. pp. 221–241. No. 179-180 in Langages (2010)
31. Tesnière, L.: Eléments de syntaxe structurale. Klincksieck, Paris (1959)